第1部

第2章

関数やローカル変数はCPUでどう実現されている？

C言語がCPUの違いを抽象化するしくみ

内田 公太

リスト1　同じC言語プログラムでもCPUが違うと機械語（アセンブリ言語）が変わる

```
int add(int a, int b) {
  a += b;
  return a;
}
```

（a）C言語ソースコード

```
8b 45 f8    mov  eax, DWORD PTR [rbp-0x8] ; b を eax にロード
01 45 fc    add  DWORD PTR [rbp-0x4], eax ; a に eax を足し込む
```

機械語　アセンブリ言語

（b）x86-64アーキテクチャの機械語とアセンブリ言語

```
b9400be9    ldr  w9, [sp, #8]  ; b を w9 にロード
b9400fe8    ldr  w8, [sp, #12] ; a を w8 にロード
0b090108    add  w8, w8, w9    ; w8 に w9 を足し込む
b9000fe8    str  w8, [sp, #12] ; w8 を a にストア
```

機械語　アセンブリ言語

（c）AArch64アーキテクチャの機械語とアセンブリ言語

CPUにより関数やローカル変数などの実現方法は異なります．しかし，C言語のレベルでは同じ書き方で問題ありません．これはC言語の抽象化能力のおかげです．

本章では，C言語がCPUの違いを抽象化するしくみを説明します．具体的には，あるソースコードが異なるCPUに対してどのようにコンパイルされるのかを見てみます．

また，本章以降では，具体的なレジスタ名や命令が数多く登場しますが，事前に詳しく知っておく必要はありません．

CPUが実際に実行するコードは機械語

● C言語をコンパイルして機械語に変換

C言語がCPUの違いを抽象化する例として，**リスト1**（**a**）のプログラムを考えます．変数aに変数bの値を足すだけの非常にシンプルなC言語プログラムです．2つの変数はどちらもint型です．

通常，このプログラムを実行するには，コンパイルを行って機械語にします．例えばx86-64アーキテクチャではa += bに相当する部分が**リスト1**（**b**）のような機械語になります[注1]．

機械語は単なる数値列[注2]であり，そのままでは理解しにくいので，対応するアセンブリ言語と日本語での説明を示します．

● 機械語（アセンブリ言語コード）を読んでみる

機械語の構造を簡単に説明します．第1部Appendix 2では，アセンブリ言語と機械語について，命令やレジスタなども含めてもう少し踏み込んで解説しています．

リスト1（**b**）のADD命令（2行目）に注目したとき，機械語は01 45 fcの3バイトです．1バイト目はOPコードと呼ばれるもので，命令の種類（ADD r/m32，r32）を表します．

2バイト目はModR/Mと呼ばれるもので，2つのオペランドを指定します．ModR/M = 0x45を分解するとMod = 1，RM = 5，REG = 0となります．これはオペランドの1つが［RBP（ベース・ポインタ）+ disp8］で，もう1つがEAXレジスタであることを示します．［RBP + disp8］はメモリを表す書き方で，RBPレジスタにdisp8を足したアドレスで指し示されるメモリを意味します．

3バイト目はディスプレースメントと呼ばれるもの

注1：GCCを使って，最適化を無効（-O0）でコンパイルしました．

注2：8bやf8が数字に見えないかもしれませんが，れっきとした16進数表記の数字です．ちなみに8b 45 f8の3バイトを10進数で表すと139 69 248となります．